RF and GBDT on Donors Choose Dataset

In [1]:

```
# Importing all the necessary libraries and packages

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import pickle
import os

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

from nltk.stem.porter import PorterStemmer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

from tqdm import tqdm
from chart_studio import plotly  #Importing plotly from chart_studio as plotly is deprecated
according to jupyter
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading Data

In [2]:

```
# Importing data with pandas

project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')

print("Number of data points in train data", project_data.shape)
print('\n', '-'*50, '\n')
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)

 --------------------------------------------------

The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [3]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]
project_data.head()
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	I
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	I

In [4]:

```
# Printing total no. of data points in Resource Data and the features it have.

print("Number of data points in resource data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head()
```

```
Number of data points in resource data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

Preprocessing project_subject_categories

In [5]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

catogories = list(project_data['project_subject_categories'].values)
cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

Preprocessing project_subject_subcategories

In [6]:

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_catogories = list(project_data['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

Preprocessing of Project_grade_category

In [7]:

```
# Preprocessing Project_grade_category
# Removing Special characters and 'Grade' word to make this category ready for the vectorization

sub_grade = list(project_data['project_grade_category'].values)
grade_cat_list = []
for i in sub_grade:
```

```
    for j in i.split(' '):
        j=j.replace('Grades','')
        j=j.replace('-', '_')
    grade_cat_list.append(j.lower().strip())

project_data['clean_grade_category'] = grade_cat_list
project_data.drop(['project_grade_category'], axis=1, inplace=True)

my_counter = Counter()
for word in project_data['clean_grade_category'].values:
    my_counter.update(word.split())

sub_grade_cat_dict = dict(my_counter)
sorted_sub_grade_cat_dict = dict(sorted(sub_grade_cat_dict.items(), key=lambda kv: kv[1]))
```

In [8]:

```
# Printing top values to see the changes and our updated data

project_data.head()
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	Having a class of 24 students comes with diver...	I
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	I
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	W

Merging Project_essay

In [9]:

```
# Merging all the subcategory of project_data into one category

project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)

# Printing top values to see the updated Data
project_data.head()
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pr
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	Having a class of 24 students comes with diver...	I
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	I i
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	W

In [10]:

```
# Merging price from resource_data to project_data before splitiing the data

price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')

# 'techer_prefix' has some missing values so we're filling it with the most common value which is
'Mrs.'
project_data["teacher_prefix"].fillna("Mrs.", inplace= True)
```

Splitting Data in train, CV and test data

In [11]:

```
# I have divided my train, cv and test in 60:25:25
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify=project_data['project_is_approved'])
```

In [12]:

```
# Printing no. of total values my Train, Cv and Test data have

print(y_train.value_counts())
print(y_test.value_counts())
```

```
1    62113
0    11083
Name: project_is_approved, dtype: int64
1    30593
0     5459
Name: project_is_approved, dtype: int64
```

Observations

- As we can see that we have an imbalance dataset and that leads to the failure of Decision tree
- So to avoid this problem we need to perform upsampling

Upsampling the data

In [13]:

```
# Dividing data into majority and minority so that we can upsample minority class

majority_data = X_train[X_train.project_is_approved==1]
minority_data = X_train[X_train.project_is_approved==0]
```

In [14]:

```
from sklearn.utils import resample

minority_data_upsampled = resample(minority_data, replace=True, n_samples=len(majority_data),
random_state=10)
X_train = pd.concat([majority_data, minority_data_upsampled])

# After applying Upsampling checking and printing total no. of datapoints for each class (i.e 0 an
d 1 class)
X_train.project_is_approved.value_counts()
```

Out[14]:

```
1    62113
0    62113
Name: project_is_approved, dtype: int64
```

In [15]:

```
# Updating y_train according to the upsampled data

y_train = X_train.project_is_approved
print(y_train.value_counts())
```

```
1    62113
0    62113
Name: project_is_approved, dtype: int64
```

In [16]:

```
# Dropping 'project_is_approved' column form cv and test data

X_train_pos = X_train[X_train['project_is_approved'] == 1]
X_train_neg = X_train[X_train['project_is_approved'] == 0]

X_train.drop(["project_is_approved"], axis = 1, inplace = True)
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
```

Preparing Data for model

Text preprocessing for Train, CV and Train Data

Preprocessing of Project_essay

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039

def decontracted(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
```

```
    return phrase
```

In [18]:

```
# https://gist.github.com/sebleier/554280

stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [19]:

```
# Preprocessing Project_essay on Train_data

train_preprocessed_essays = []

for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 124226/124226
[02:27<00:00, 840.41it/s]
```

In [20]:

```
# Preprocessing Project_essay on Test data

test_preprocessed_essays = []

for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 36052/36052 [00:
43<00:00, 819.70it/s]
```

Preprocessing Project_title

In [21]:

```
def decontracted2(phrase):
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\s'", "s", phrase)
    return phrase
```

In [22]:

```
# Preprocessing Project_title on Train_data

train_preprocessed_title = []
for title in tqdm(X_train['project_title'].values):
    sent = decontracted2(title)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    train_preprocessed_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 124226/124226
[00:04<00:00, 25200.15it/s]
```

In [23]:

```
# Preprocessing Project_title on Test_data

test_preprocessed_title = []
for title in tqdm(X_test['project_title'].values):
    sent = decontracted2(title)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 19976.00it/s]
```

Vectorizing Categorical Data

Response coding on clean_categories

In [24]:

```
clean_categories_pos = {}
clean_categories_total = {}

for i in X_train_pos['clean_categories']:
    for j in i.split():
        if j not in clean_categories_pos:
            clean_categories_pos[j] = 1
            clean_categories_total[j] = 1
        else:
            clean_categories_pos[j] += 1
            clean_categories_total[j] += 1
```

In [25]:

```
clean_categories_neg = {}

for i in X_train_neg['clean_categories']:
```

```
    for j in i.split():
        if j not in clean_categories_neg:
            clean_categories_neg[j] = 1
        else:
            clean_categories_neg[j] += 1
        clean_categories_total[j] += 1
```

In [26]:

```
categories_prob_pos = {}
categories_prob_neg = {}

for i in clean_categories_total.keys():
    categories_prob_pos[i] = round(((clean_categories_pos[i]) / float(clean_categories_total[i])),
2)

for i in clean_categories_total.keys():
    categories_prob_neg[i] = round(((clean_categories_neg[i]) / float(clean_categories_total[i])),
2)
```

In [27]:

```
train_clean_cat_pos_score = []
train_clean_cat_neg_score = []

for i in X_train["clean_categories"] :
    j = i.split()
    if len(j) == 1 :
        train_clean_cat_neg_score.append(categories_prob_neg[i])
        train_clean_cat_pos_score.append(categories_prob_pos[i])
    else :
        if len(j) == 2:
            a0 = categories_prob_neg[j[0]]
            b0 = categories_prob_neg[j[1]]
            a1 = categories_prob_pos[j[0]]
            b1 = categories_prob_pos[j[1]]
            train_clean_cat_neg_score.append(round((a0 * b0), 2))
            train_clean_cat_pos_score.append(round((a1 * b1), 2))
        else:
            a0 = categories_prob_neg[j[0]]
            b0 = categories_prob_neg[j[1]]
            c0 = categories_prob_neg[j[2]]
            a1 = categories_prob_pos[j[0]]
            b1 = categories_prob_pos[j[1]]
            c1 = categories_prob_pos[j[2]]
            train_clean_cat_neg_score.append(round((a0 * b0 * c0), 2))
            train_clean_cat_pos_score.append(round((a1 * b1 * c1), 2))

X_train["train_clean_cat_pos_score"] = train_clean_cat_pos_score
X_train["train_clean_cat_neg_score"] = train_clean_cat_neg_score

train_clean_cat_pos_score = X_train["train_clean_cat_pos_score"].values.reshape(-1, 1)
train_clean_cat_neg_score = X_train["train_clean_cat_neg_score"].values.reshape(-1, 1)
```

In [28]:

```
test_clean_cat_pos_score = []
test_clean_cat_neg_score = []

for i in X_test["clean_categories"] :
    j = i.split()
    if len(j) == 1 :
        test_clean_cat_neg_score.append(categories_prob_neg[i])
        test_clean_cat_pos_score.append(categories_prob_pos[i])
    else :
        if len(j) == 2:
            a0 = categories_prob_neg[j[0]]
            b0 = categories_prob_neg[j[1]]
            a1 = categories_prob_pos[j[0]]
            b1 = categories_prob_pos[j[1]]
            test_clean_cat_neg_score.append(round((a0 * b0), 2))
            test_clean_cat_pos_score.append(round((a1 * b1), 2))
        else:
            a0 = categories_prob_neg[j[0]]
```

```
            b0 = categories_prob_neg[j[1]]
            c0 = categories_prob_neg[j[2]]
            a1 = categories_prob_pos[j[0]]
            b1 = categories_prob_pos[j[1]]
            c1 = categories_prob_pos[j[2]]
            test_clean_cat_neg_score.append(round((a0 * b0 * c0), 2))
            test_clean_cat_pos_score.append(round((a1 * b1 * c1), 2))

X_test["test_clean_cat_pos_score"] = test_clean_cat_pos_score
X_test["test_clean_cat_neg_score"] = test_clean_cat_neg_score

test_clean_cat_pos_score = X_test["test_clean_cat_pos_score"].values.reshape(-1,1)
test_clean_cat_neg_score = X_test["test_clean_cat_neg_score"].values.reshape(-1,1)
```

Response coding on clean_subcategories

In [29]:

```
clean_subcategories_pos = {}
clean_subcategories_total = {}

for i in X_train_pos['clean_subcategories']:
    for j in i.split():
        if j not in clean_subcategories_pos:
            clean_subcategories_pos[j] = 1
            clean_subcategories_total[j] = 1
        else:
            clean_subcategories_pos[j] += 1
            clean_subcategories_total[j] += 1
```

In [30]:

```
clean_subcategories_neg = {}

for i in X_train_neg['clean_subcategories']:
    for j in i.split():
        if j not in clean_subcategories_neg:
            clean_subcategories_neg[j] = 1
        else:
            clean_subcategories_neg[j] += 1
        clean_subcategories_total[j] += 1
```

In [31]:

```
subcategories_prob_pos = {}
subcategories_prob_neg = {}

for i in clean_subcategories_total.keys():
    subcategories_prob_pos[i] = round(((clean_subcategories_pos[i]) /
float(clean_subcategories_total[i])), 2)

for i in clean_subcategories_total.keys():
    subcategories_prob_neg[i] = round(((clean_subcategories_neg[i]) /
float(clean_subcategories_total[i])), 2)
```

In [32]:

```
train_clean_subcat_pos_score = []
train_clean_subcat_neg_score = []

for i in X_train["clean_subcategories"] :
    j = i.split()
    if len(j) == 1 :
        train_clean_subcat_neg_score.append(subcategories_prob_neg[i])
        train_clean_subcat_pos_score.append(subcategories_prob_pos[i])
    else :
        if len(j) == 2:
            a0 = subcategories_prob_neg[j[0]]
            b0 = subcategories_prob_neg[j[1]]
            a1 = subcategories_prob_pos[j[0]]
            b1 = subcategories_prob_pos[j[1]]
```

```
            train_clean_subcat_neg_score.append(round((a0 * b0), 2))
            train_clean_subcat_pos_score.append(round((a1 * b1), 2))
        else:
            a0 = subcategories_prob_neg[j[0]]
            b0 = subcategories_prob_neg[j[1]]
            c0 = subcategories_prob_neg[j[2]]
            a1 = subcategories_prob_pos[j[0]]
            b1 = subcategories_prob_pos[j[1]]
            c1 = subcategories_prob_pos[j[2]]
            train_clean_subcat_neg_score.append(round((a0 * b0 * c0), 2))
            train_clean_subcat_pos_score.append(round((a1 * b1 * c1), 2))

X_train["train_clean_subcat_pos_score"] = train_clean_subcat_pos_score
X_train["train_clean_subcat_neg_score"] = train_clean_subcat_neg_score

train_clean_subcat_pos_score = X_train["train_clean_subcat_pos_score"].values.reshape(-1, 1)
train_clean_subcat_neg_score = X_train["train_clean_subcat_neg_score"].values.reshape(-1, 1)
```

In [33]:

```
test_clean_subcat_pos_score = []
test_clean_subcat_neg_score = []

for i in X_test["clean_subcategories"] :
    j = i.split()
    if len(j) == 1 :
        test_clean_subcat_neg_score.append(subcategories_prob_neg[i])
        test_clean_subcat_pos_score.append(subcategories_prob_pos[i])
    else :
        if len(j) == 2:
            a0 = subcategories_prob_neg[j[0]]
            b0 = subcategories_prob_neg[j[1]]
            a1 = subcategories_prob_pos[j[0]]
            b1 = subcategories_prob_pos[j[1]]
            test_clean_subcat_neg_score.append(round((a0 * b0), 2))
            test_clean_subcat_pos_score.append(round((a1 * b1), 2))
        else:
            a0 = subcategories_prob_neg[j[0]]
            b0 = subcategories_prob_neg[j[1]]
            c0 = subcategories_prob_neg[j[2]]
            a1 = subcategories_prob_pos[j[0]]
            b1 = subcategories_prob_pos[j[1]]
            c1 = subcategories_prob_pos[j[2]]
            test_clean_subcat_neg_score.append(round((a0 * b0 * c0), 2))
            test_clean_subcat_pos_score.append(round((a1 * b1 * c1), 2))

X_test["test_clean_subcat_pos_score"] = test_clean_subcat_pos_score
X_test["test_clean_subcat_neg_score"] = test_clean_subcat_neg_score

test_clean_subcat_pos_score = X_test["test_clean_subcat_pos_score"].values.reshape(-1,1)
test_clean_subcat_neg_score = X_test["test_clean_subcat_neg_score"].values.reshape(-1,1)
```

Response coding on school_state

In [34]:

```
school_state_pos = {}
school_state_total = {}

for i in X_train_pos['school_state']:
    for j in i.split():
        if j not in school_state_pos:
            school_state_pos[j] = 1
            school_state_total[j] = 1
        else:
            school_state_pos[j] += 1
            school_state_total[j] += 1
```

In [35]:

```
school_state_neg = {}
```

```
for i in X_train_neg['school_state']:
    for j in i.split():
        if j not in school_state_neg:
            school_state_neg[j] = 1
        else:
            school_state_neg[j] += 1
        school_state_total[j] += 1
```

In [36]:

```
school_state_prob_pos = {}
school_state_prob_neg = {}

for i in school_state_total.keys():
    school_state_prob_pos[i] = round(((school_state_pos[i]) / float(school_state_total[i])), 2)

for i in school_state_total.keys():
    school_state_prob_neg[i] = round(((school_state_neg[i]) / float(school_state_total[i])), 2)
```

In [37]:

```
train_school_state_pos_score = []
train_school_state_neg_score = []

for i in X_train["school_state"]:
    train_school_state_pos_score.append(school_state_prob_pos[i])
    train_school_state_neg_score.append(school_state_prob_neg[i])

X_train["train_school_state_pos_score"] = train_school_state_pos_score
X_train["train_school_state_neg_score"] = train_school_state_neg_score

train_school_state_pos_score = X_train["train_school_state_pos_score"].values.reshape(-1,1)
train_school_state_neg_score = X_train["train_school_state_neg_score"].values.reshape(-1,1)
```

In [38]:

```
test_school_state_pos_score = []
test_school_state_neg_score = []

for i in X_test["school_state"]:
    test_school_state_pos_score.append(school_state_prob_pos[i])
    test_school_state_neg_score.append(school_state_prob_neg[i])

X_test["test_school_state_pos_score"] = test_school_state_pos_score
X_test["test_school_state_neg_score"] = test_school_state_neg_score

test_school_state_pos_score = X_test["test_school_state_pos_score"].values.reshape(-1,1)
test_school_state_neg_score = X_test["test_school_state_neg_score"].values.reshape(-1,1)
```

Response coding on teacher_prefix

In [39]:

```
teacher_prefix_pos = {}
teacher_prefix_total = {}

for i in X_train_pos['teacher_prefix']:
    for j in i.split():
        if j not in teacher_prefix_pos:
            teacher_prefix_pos[j] = 1
            teacher_prefix_total[j] = 1
        else:
            teacher_prefix_pos[j] += 1
            teacher_prefix_total[j] += 1
```

In [40]:

```
teacher_prefix_neg = {}

for i in X_train_neg['teacher_prefix']:
```

```
    for j in i.split():
        if j not in teacher_prefix_neg:
            teacher_prefix_neg[j] = 1
        else:
            teacher_prefix_neg[j] += 1
        teacher_prefix_total[j] += 1
```

In [41]:

```
teacher_prefix_prob_pos = {}
teacher_prefix_prob_neg = {}

for i in teacher_prefix_total.keys():
    teacher_prefix_prob_pos[i] = round(((teacher_prefix_pos[i]) / float(teacher_prefix_total[i])),
2)

for i in teacher_prefix_total.keys():
    teacher_prefix_prob_neg[i] = round(((teacher_prefix_neg[i]) / float(teacher_prefix_total[i])),
2)
```

In [42]:

```
train_teacher_prefix_pos_score = []
train_teacher_prefix_neg_score = []

for i in X_train["teacher_prefix"]:
    train_teacher_prefix_pos_score.append(teacher_prefix_prob_pos[i])
    train_teacher_prefix_neg_score.append(teacher_prefix_prob_neg[i])

X_train["train_teacher_prefix_pos_score"] = train_teacher_prefix_pos_score
X_train["train_teacher_prefix_neg_score"] = train_teacher_prefix_neg_score

train_teacher_prefix_pos_score = X_train["train_teacher_prefix_pos_score"].values.reshape(-1, 1)
train_teacher_prefix_neg_score = X_train["train_teacher_prefix_neg_score"].values.reshape(-1, 1)
```

In [43]:

```
test_teacher_prefix_pos_score = []
test_teacher_prefix_neg_score = []

for i in X_test["teacher_prefix"]:
    test_teacher_prefix_pos_score.append(teacher_prefix_prob_pos[i])
    test_teacher_prefix_neg_score.append(teacher_prefix_prob_neg[i])

X_test["test_teacher_prefix_pos_score"] = test_teacher_prefix_pos_score
X_test["test_teacher_prefix_neg_score"] = test_teacher_prefix_neg_score

test_teacher_prefix_pos_score = X_test["test_teacher_prefix_pos_score"].values.reshape(-1, 1)
test_teacher_prefix_neg_score = X_test["test_teacher_prefix_neg_score"].values.reshape(-1, 1)
```

Response coding on clean_grade_category

In [44]:

```
clean_grade_pos = {}
clean_grade_total = {}

for i in X_train_pos['clean_grade_category']:
    for j in i.split():
        if j not in clean_grade_pos:
            clean_grade_pos[j] = 1
            clean_grade_total[j] = 1
        else:
            clean_grade_pos[j] += 1
            clean_grade_total[j] += 1
```

In [45]:

```
clean_grade_neg = {}
```

```
for i in X_train_neg['clean_grade_category']:
    for j in i.split():
        if j not in clean_grade_neg:
            clean_grade_neg[j] = 1
        else:
            clean_grade_neg[j] += 1
        clean_grade_total[j] += 1
```

In [46]:

```
clean_grade_prob_pos = {}
clean_grade_prob_neg = {}

for i in clean_grade_total.keys():
    clean_grade_prob_pos[i] = round(((clean_grade_pos[i]) / float(clean_grade_total[i])), 2)

for i in clean_grade_total.keys():
    clean_grade_prob_neg[i] = round(((clean_grade_neg[i]) / float(clean_grade_total[i])), 2)
```

In [47]:

```
train_clean_grade_pos_score = []
train_clean_grade_neg_score = []

for i in X_train["clean_grade_category"]:
    train_clean_grade_pos_score.append(clean_grade_prob_pos[i])
    train_clean_grade_neg_score.append(clean_grade_prob_neg[i])

X_train["train_clean_grade_pos_score"] = train_clean_grade_pos_score
X_train["train_clean_grade_neg_score"] = train_clean_grade_neg_score

train_clean_grade_pos_score = X_train["train_clean_grade_pos_score"].values.reshape(-1, 1)
train_clean_grade_neg_score = X_train["train_clean_grade_neg_score"].values.reshape(-1, 1)
```

In [48]:

```
test_clean_grade_pos_score = []
test_clean_grade_neg_score = []

for i in X_test["clean_grade_category"]:
    test_clean_grade_pos_score.append(clean_grade_prob_pos[i])
    test_clean_grade_neg_score.append(clean_grade_prob_neg[i])

X_test["test_clean_grade_pos_score"] = test_clean_grade_pos_score
X_test["test_clean_grade_neg_score"] = test_clean_grade_neg_score

test_clean_grade_pos_score = X_test["test_clean_grade_pos_score"].values.reshape(-1, 1)
test_clean_grade_neg_score = X_test["test_clean_grade_neg_score"].values.reshape(-1, 1)
```

Bag of Words on Project Essay for train and test data

In [49]:

```
vectorizer = CountVectorizer(min_df = 10, max_features=5000)
vectorizer.fit(train_preprocessed_essays)

X_train_essay_bow = vectorizer.transform(train_preprocessed_essays)
X_test_essay_bow = vectorizer.transform(test_preprocessed_essays)
```

In [50]:

```
print("Shape of train_matrix after BoW on project_essay : ", X_train_essay_bow.shape,
y_train.shape)
print("\nShape of test_matrix after BoW on project_essay : ", X_test_essay_bow.shape, y_test.shape
)
```

```
Shape of train_matrix after BoW on project_essay :  (124226, 5000) (124226,)

Shape of test_matrix after BoW on project_essay :  (36052, 5000) (36052,)
```

Bag of Words on Project title for train and test data

In [51]:

```
vectorizer = CountVectorizer(min_df=6, max_features=5000)
vectorizer.fit(train_preprocessed_title)

X_train_title_bow = vectorizer.transform(train_preprocessed_title)
X_test_title_bow = vectorizer.transform(test_preprocessed_title)
```

In [52]:

```
print("Shape of train_matrix after BoW on project_title : ", X_train_title_bow.shape,
y_train.shape)
print("\nShape of test_matrix after BoW on project_title : ", X_test_title_bow.shape, y_test.shape
)
```

```
Shape of train_matrix after BoW on project_title :  (124226, 5000) (124226,)

Shape of test_matrix after BoW on project_title :  (36052, 5000) (36052,)
```

Tf-IDF Vectorizer on preprocessed_essays for train and test data

In [53]:

```
vectorizer = TfidfVectorizer(min_df = 10, max_features=5000)
vectorizer.fit(train_preprocessed_essays)

X_train_essay_tf = vectorizer.transform(train_preprocessed_essays)
X_test_essay_tf = vectorizer.transform(test_preprocessed_essays)
```

In [54]:

```
print("Shape of train_matrix after tfidf on project_essay : ", X_train_essay_tf.shape,
y_train.shape)
print("\nShape of test_matrix after tfidf on project_essay : ", X_test_essay_tf.shape,
y_test.shape)
```

```
Shape of train_matrix after tfidf on project_essay :  (124226, 5000) (124226,)

Shape of test_matrix after tfidf on project_essay :  (36052, 5000) (36052,)
```

Tf-IDF Vectorizer on preprocessed_title for train and test data

In [55]:

```
vectorizer = TfidfVectorizer(min_df=6, max_features=5000)
vectorizer.fit(train_preprocessed_title)

X_train_title_tf = vectorizer.transform(train_preprocessed_title)
X_test_title_tf = vectorizer.transform(test_preprocessed_title)
```

In [56]:

```
print("Shape of train_matrix after tfidf on project_title : ", X_train_title_tf.shape,
y_train.shape)
print("\nShape of test_matrix after tfidf on project_title : ", X_test_title_tf.shape,
y_test.shape)
```

```
Shape of train_matrix after tfidf on project_title :  (124226, 5000) (124226,)

Shape of test_matrix after tfidf on project_title :  (36052, 5000) (36052,)
```

Avg W2V on preprocessed_essay for train and test data

In [57]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

In [58]:

```
# Avg W2V on Project_essay for Train Data

X_train_essay_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_essay_avgW2V.append(vector)

print(len(X_train_essay_avgW2V))
print(len(X_train_essay_avgW2V[0]))
```

```
100%|██████████████████████████████████████████████████| 124226/124226
[01:02<00:00, 1972.34it/s]
```

```
124226
300
```

In [59]:

```
# Avg W2V on Project_essay for Test Data

X_test_essay_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avgW2V.append(vector)
```

```
100%|██████████████████████████████████████████████████| 36052/36052
[00:19<00:00, 1817.62it/s]
```

Avg W2V on preprocessed_title for train and test data

In [60]:

```
# Avg W2V on Project_title for Train Data

X_train_title_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(train_preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
```

```
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_title_avgW2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 124226/124226
[00:02<00:00, 45235.39it/s]
```

In [61]:

```
# Avg W2V on Project_title for Test Data

X_test_title_avgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_title_avgW2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 50746.62it/s]
```

Tf-idf weighted W2V on preprocessed_essay for train and test data

In [62]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [63]:

```
# Tf-Idf W2V on Project_essay for Train Data

X_train_essay_tfidf_W2V = []

for sentence in tqdm(train_preprocessed_essays):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_W2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 124226/124226
[04:51<00:00, 425.67it/s]
```

In [64]:

```
# Tf-Idf W2V on Project_essay for Test Data

X_test_essay_tfidf_W2V = []

for sentence in tqdm(test_preprocessed_essays):
    vector = np.zeros(300)
```

```
    tf_idf_weight =0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_W2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 36052/36052 [01:
25<00:00, 421.92it/s]
```

Tf-idf weighted W2V on preprocessed_title for train and test data

In [65]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_title)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [66]:

```
# Tf-Idf W2V on Project_title for Train Data

X_train_title_tfidf_W2V = []

for sentence in tqdm(train_preprocessed_title):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_title_tfidf_W2V.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 124226/124226
[00:05<00:00, 21231.73it/s]
```

In [67]:

```
# Tf-Idf W2V on Project_title for Test Data

X_test_title_tfidf_W2V = []

for sentence in tqdm(test_preprocessed_title):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_title_tfidf_W2V.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 20876.75it/s]
```

Vectorizing Numerical Features

In [68]:

```
# Vectorizing Price Feature on Train, CV and Test data
from sklearn.preprocessing import MinMaxScaler

price_scalar = MinMaxScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = price_scalar.transform(X_train['price'].values.reshape(-1,1))
X_test_price_std = price_scalar.transform(X_test['price'].values.reshape(-1,1))

# Printing train data after applying minmaxscaler to see the changes
print(X_train_price_std)
```

```
[[0.02550123]
 [0.05469008]
 [0.05942286]
 ...
 [0.08258571]
 [0.02511417]
 [0.02993597]]
```

In [69]:

```
print("Printing shape of Train and Test data after vectorizing price")

print(X_train_price_std.shape, y_train.shape)
print(X_test_price_std.shape, y_test.shape)
```

```
Printing shape of Train and Test data after vectorizing price
(124226, 1) (124226,)
(36052, 1) (36052,)
```

In [70]:

```
# Vectorizing teacher_number_of_previously_posted_projects on Train, CV and test data

previously_posted_projects_scalar = MinMaxScaler()
previously_posted_projects_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1,1))

X_train_posted_projects_std =
previously_posted_projects_scalar.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
X_test_posted_projects_std =
previously_posted_projects_scalar.transform(X_test['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))

print(X_train_posted_projects_std)
```

```
[[0.00221729]
 [0.04656319]
 [0.02439024]
 ...
 [0.        ]
 [0.00221729]
 [0.00443459]]
```

In [71]:

```
print("Printing shape of Train and Test data after vectorizing
teacher_number_of_previously_posted_projects")

print(X_train_posted_projects_std.shape, y_train.shape)
print(X_test_posted_projects_std.shape, y_test.shape)
```

```
Printing shape of Train and Test data after vectorizing
```

```
teacher_number_of_previously_posted_projects
(124226, 1) (124226,)
(36052, 1) (36052,)
```

Merging all the features

In [72]:

```
# Before we merge all the features we need to convert avg_W2V and tf-idf_avgW2V list to ndarray

# Coverting avgW2V of essays into ndarray
X_train_essay_avgW2V = np.array(X_train_essay_avgW2V)
X_test_essay_avgW2V = np.array(X_test_essay_avgW2V)

# Coverting avgW2V of title into ndarray
X_train_title_avgW2V = np.array(X_train_title_avgW2V)
X_test_title_avgW2V = np.array(X_test_title_avgW2V)

# Coverting tf-Idf_avgW2V of essays into ndarray
X_train_essay_tfidf_W2V = np.array(X_train_essay_tfidf_W2V)
X_test_essay_tfidf_W2V = np.array(X_test_essay_tfidf_W2V)

# Coverting tf-Idf_avgW2V of title into ndarray
X_train_title_tfidf_W2V = np.array(X_train_title_tfidf_W2V)
X_test_title_tfidf_W2V = np.array(X_test_title_tfidf_W2V)
```

In [73]:

```
# Merging all the features for Set-1
from scipy.sparse import hstack

X_train_s1 = hstack((X_train_essay_bow, X_train_title_bow, X_train_posted_projects_std,
X_train_price_std, train_clean_grade_neg_score, train_teacher_prefix_neg_score,
train_school_state_neg_score, train_clean_subcat_neg_score, train_clean_cat_neg_score,
train_clean_grade_pos_score, train_teacher_prefix_pos_score, train_school_state_pos_score,
train_clean_subcat_pos_score, train_clean_cat_pos_score)).tocsr()
X_test_s1 = hstack((X_test_essay_bow, X_test_title_bow, X_test_posted_projects_std,
X_test_price_std, test_clean_grade_neg_score, test_teacher_prefix_neg_score,
test_school_state_neg_score, test_clean_subcat_neg_score, test_clean_cat_neg_score,
test_clean_grade_pos_score, test_teacher_prefix_pos_score, test_school_state_pos_score,
test_clean_subcat_pos_score, test_clean_cat_pos_score)).tocsr()

print("Final Data matrix of Set-1\n")
print(X_train_s1.shape, y_train.shape)
print(X_test_s1.shape, y_test.shape)
```

```
Final Data matrix of Set-1

(124226, 10012) (124226,)
(36052, 10012) (36052,)
```

In [74]:

```
# Merging all the features for Set-2

X_train_s2 = hstack((X_train_essay_tf, X_train_title_tf, X_train_posted_projects_std,
X_train_price_std, train_clean_grade_neg_score, train_teacher_prefix_neg_score,
train_school_state_neg_score, train_clean_subcat_neg_score, train_clean_cat_neg_score,
train_clean_grade_pos_score, train_teacher_prefix_pos_score, train_school_state_pos_score,
train_clean_subcat_pos_score, train_clean_cat_pos_score)).tocsr()
X_test_s2 = hstack((X_test_essay_tf, X_test_title_tf, X_test_posted_projects_std, X_test_price_std
, test_clean_grade_neg_score, test_teacher_prefix_neg_score, test_school_state_neg_score,
test_clean_subcat_neg_score, test_clean_cat_neg_score, test_clean_grade_pos_score,
test_teacher_prefix_pos_score, test_school_state_pos_score, test_clean_subcat_pos_score,
test_clean_cat_pos_score)).tocsr()

print("Final Data matrix of Set-2\n")
print(X_train_s2.shape, y_train.shape)
print(X_test_s2.shape, y_test.shape)
```

```
Final Data matrix of Set-2
```

```
(124226, 10012) (124226,)
(36052, 10012) (36052,)
```

In [75]:

```
# Merging all the features for Set-3
from scipy.sparse import csr_matrix

X_train_s3 = hstack((X_train_posted_projects_std, X_train_price_std, train_clean_grade_neg_score,
train_teacher_prefix_neg_score, train_school_state_neg_score, train_clean_subcat_neg_score,
train_clean_cat_neg_score, train_clean_grade_pos_score, train_teacher_prefix_pos_score,
train_school_state_pos_score, train_clean_subcat_pos_score, train_clean_cat_pos_score,
X_train_essay_avgW2V, csr_matrix(X_train_title_avgW2V))).tocsr()
X_test_s3 = hstack((X_test_essay_avgW2V, csr_matrix(X_test_title_avgW2V),
X_test_posted_projects_std, X_test_price_std, test_clean_grade_neg_score,
test_teacher_prefix_neg_score, test_school_state_neg_score, test_clean_subcat_neg_score,
test_clean_cat_neg_score, test_clean_grade_pos_score, test_teacher_prefix_pos_score,
test_school_state_pos_score, test_clean_subcat_pos_score, test_clean_cat_pos_score)).tocsr()

print("Final Data matrix of Set-3\n")
print(X_train_s3.shape, y_train.shape)
print(X_test_s3.shape, y_test.shape)
```

```
Final Data matrix of Set-3

(124226, 612) (124226,)
(36052, 612) (36052,)
```

In [76]:

```
# Merging all the features for Set-4

X_train_s4 = hstack((X_train_essay_tfidf_W2V, csr_matrix(X_train_title_tfidf_W2V),
X_train_posted_projects_std, X_train_price_std, train_clean_grade_neg_score,
train_teacher_prefix_neg_score, train_school_state_neg_score, train_clean_subcat_neg_score,
train_clean_cat_neg_score, train_clean_grade_pos_score, train_teacher_prefix_pos_score,
train_school_state_pos_score, train_clean_subcat_pos_score, train_clean_cat_pos_score)).tocsr()
X_test_s4 = hstack((X_test_essay_tfidf_W2V, csr_matrix(X_test_title_tfidf_W2V),
X_test_posted_projects_std, X_test_price_std, test_clean_grade_neg_score,
test_teacher_prefix_neg_score, test_school_state_neg_score, test_clean_subcat_neg_score,
test_clean_cat_neg_score, test_clean_grade_pos_score, test_teacher_prefix_pos_score,
test_school_state_pos_score, test_clean_subcat_pos_score, test_clean_cat_pos_score)).tocsr()

print("Final Data matrix of Set-4\n")
print(X_train_s4.shape, y_train.shape)
print(X_test_s4.shape, y_test.shape)
```

```
Final Data matrix of Set-4

(124226, 612) (124226,)
(36052, 612) (36052,)
```

Applying Random Forest on Set-1(BoW)

In [77]:

```
# Selecting only 50k points to reduce the run-time and avoid memory error

X_train_s1 = X_train_s1[37113:87113]
y_train_s1 = y_train[37113:87113]

X_train_s1.shape, y_train_s1.shape
```

Out[77]:

```
((50000, 10012), (50000,))
```

In [78]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

rf1 = RandomForestClassifier(class_weight = 'balanced', min_samples_split = 10)
parameters = {'n_estimators' : [50, 100, 200, 300, 500, 1000], 'max_depth' : [2, 3, 4, 5, 6, 8, 10]}
clf1 = GridSearchCV(rf1, parameters, cv = 3, scoring = 'roc_auc')
clf1.fit(X_train_s1, y_train_s1)
```

Out[78]:

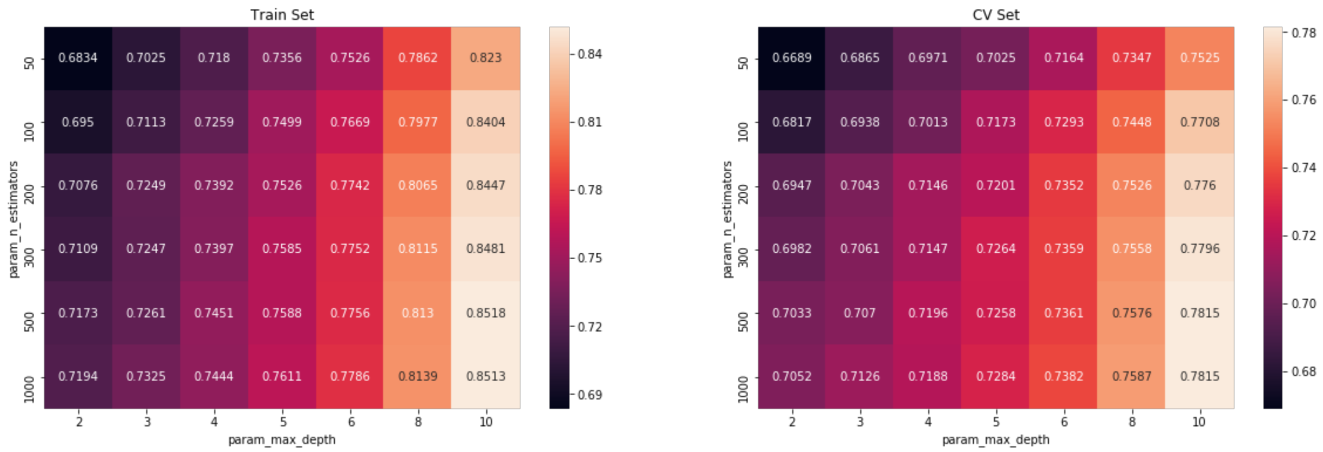
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [50, 100, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 8,
10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Plotting Heatmap for Set-1

In [79]:

```python
max_scores = pd.DataFrame(clf1.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [80]:

```python
print(clf1.best_estimator_)
print(clf1.score(X_train_s1, y_train_s1))
print(clf1.score(X_test_s1, y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.8414919696000001
0.6965916263232431
```

Observations

- Based on the heatmap we can see that values max_depth = 6 and n_estimators = 500 will give us the best results.

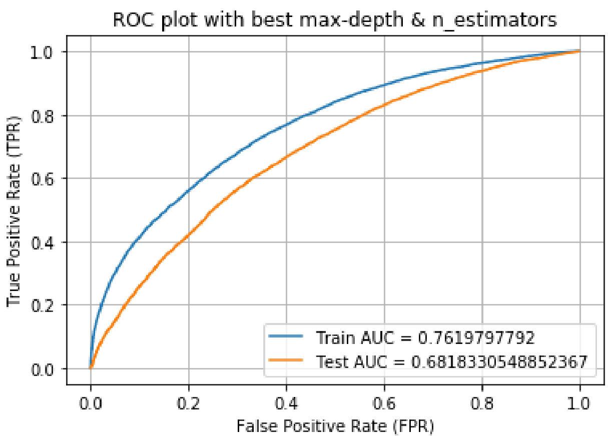
Applying Random Forest with Hyperparameter tuning on Set-1

In [81]:

```
rf = RandomForestClassifier(class_weight = 'balanced', max_depth = 6, min_samples_split = 10, n_est
imators = 500)
rf.fit(X_train_s1, y_train_s1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s1, rf.predict_proba(X_train_s1)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, rf.predict_proba(X_test_s1)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrices for Train and Test data

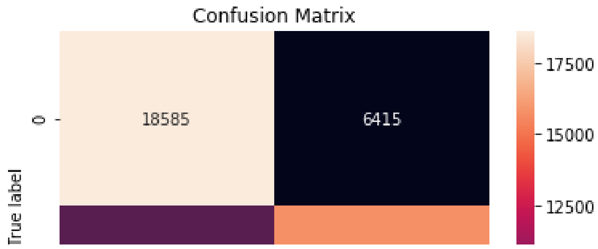
In [82]:

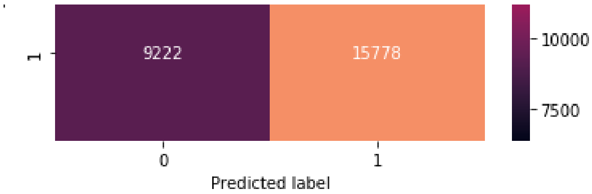
```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s1, rf.predict(X_train_s1)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[82]:

```
Text(0.5,1,'Confusion Matrix')
```





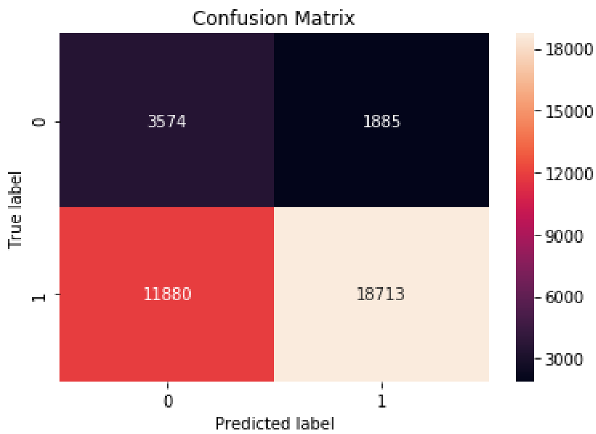
In [83]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, rf.predict(X_test_s1)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[83]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.681
- The best paramteres to use with set-1 is max_depth = 6 and n_estimators = 500
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.

Applying Random Forest on Set-2 (Tf-Idf)

In [84]:

```
# Selecting 50k points

X_train_s2 = X_train_s2[37113:87113]
y_train_s2 = y_train[37113:87113]

X_train_s2.shape, y_train_s2.shape
```

Out[84]:

```
((50000, 10012), (50000,))
```

In [85]:

```
rf2 = RandomForestClassifier(class_weight = 'balanced', min_samples_split = 10)
parameters = {'n_estimators' : [50, 100, 200, 300, 500, 1000], 'max_depth' : [2, 3, 4, 5, 6, 8, 10]}
clf2 = GridSearchCV(rf2, parameters, cv = 3, scoring = 'roc_auc')
clf2.fit(X_train_s2, y_train_s2)
```

Out[85]:

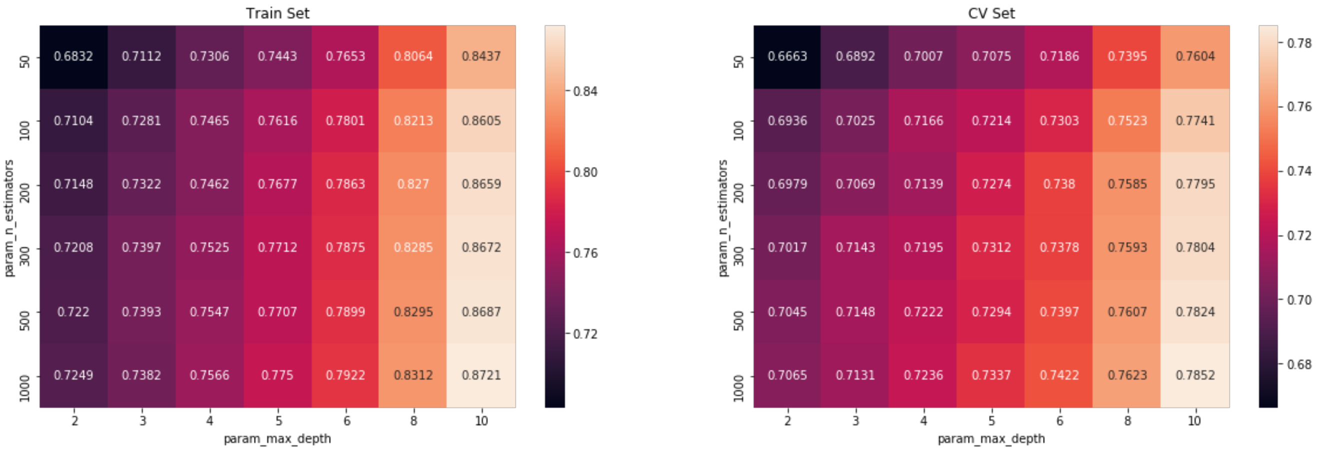
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [50, 100, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 8,
10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Heatmap for Set-2

In [86]:

```
max_scores = pd.DataFrame(clf2.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1, 2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [87]:

```
print(clf2.best_estimator_)
print(clf2.score(X_train_s2, y_train_s2))
print(clf2.score(X_test_s2, y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.8612357088
0.6990502151263704
```

Observations

- Based on the heatmap we can see that values max_depth = 6 and n_estimators = 500 will give us the best results.

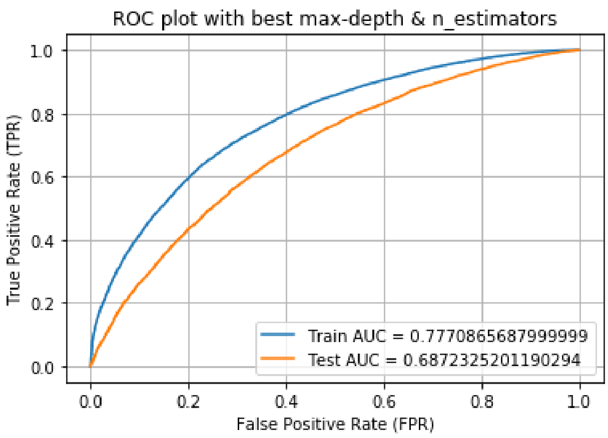
Applying Random Forest on Set-2 with Hyperparameter Tuning

In [88]:

```
rf = RandomForestClassifier(class_weight = 'balanced', max_depth = 6, min_samples_split = 10, n_est
imators = 500)
rf.fit(X_train_s2, y_train_s2)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s2, rf.predict_proba(X_train_s2)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, rf.predict_proba(X_test_s2)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion matrix for Train and Test data

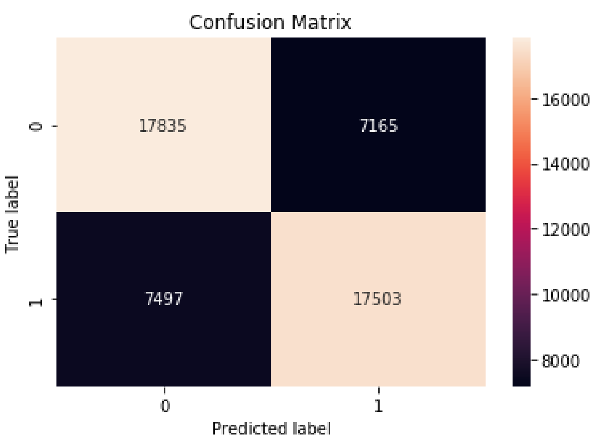
In [89]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s2, rf.predict(X_train_s2)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[89]:

```
Text(0.5,1,'Confusion Matrix')
```



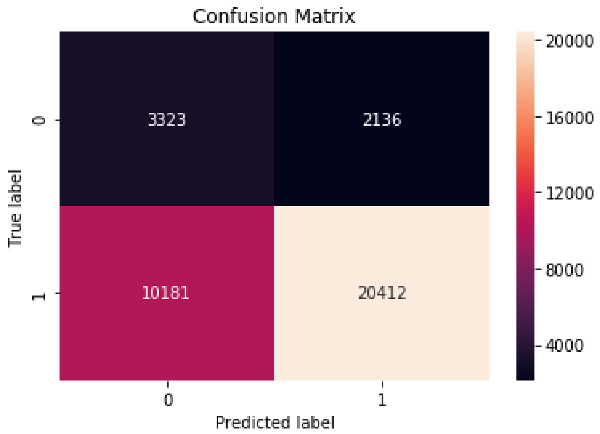
In [90]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, rf.predict(X_test_s2)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[90]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.687
- The best paramteres to use with set-2 is max_depth = 6 and n_estimators = 500
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.

Applying Random Forest on Set-3 (AvgW2V)

In [91]:

```
# Selecting 20k points

X_train_s3 = X_train_s3[52113:72113]
y_train_s3 = y_train[52113:72113]

X_test_s3 = X_test_s3[14526:21526]
y_test_s3 = y_test[14526:21526]

print(X_train_s3.shape, y_train_s3.shape)
print(X_test_s3.shape, y_test_s3.shape)
```

```
(20000, 612) (20000,)
(7000, 612) (7000,)
```

In [92]:

```
rf3 = RandomForestClassifier(class_weight = 'balanced', min_samples_split = 10)
parameters = {'n_estimators' : [50, 100, 200, 300, 500, 1000], 'max_depth' : [2, 3, 4, 5, 6, 8, 10]}
clf3 = GridSearchCV(rf3, parameters, cv = 3, scoring = 'roc_auc')
clf3.fit(X_train_s3, y_train_s3)
```

Out[92]:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
```

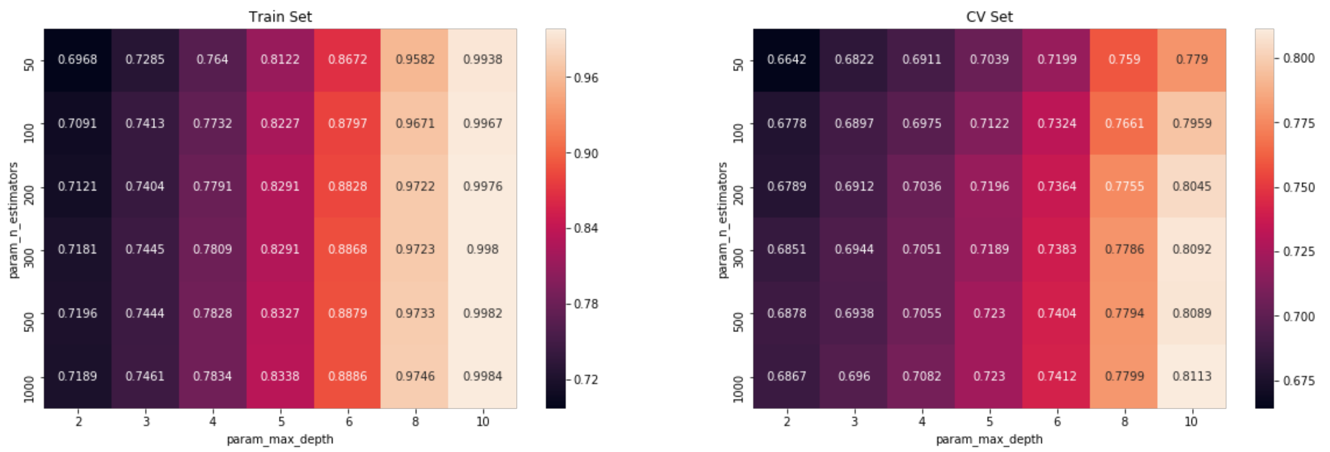
```
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [50, 100, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 8,
10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Heatmap for Set-3

In [93]:

```
max_scores = pd.DataFrame(clf3.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1, 2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [94]:

```
print(clf3.best_estimator_)
print(clf3.score(X_train_s3, y_train_s3))
print(clf3.score(X_test_s3, y_test_s3))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.99555985
0.5127866484636237
```

Observations

- Based on the heatmap we can see that values max_depth = 4 and n_estimators = 1000 will give us the best results.

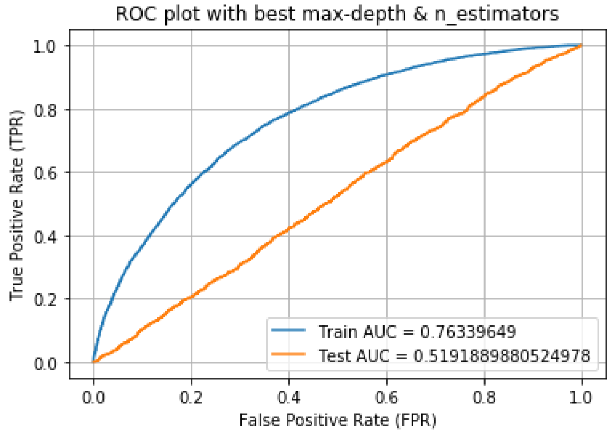
Applying Random Forest with Hyperparameter Tuning on Set-3

In [131]:

```
rf = RandomForestClassifier(class_weight = 'balanced', max_depth = 4, min_samples_split = 10, n_est
imators = 1000)
rf.fit(X_train_s3, y_train_s3)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s3, rf.predict_proba(X_train_s3)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s3, rf.predict_proba(X_test_s3)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

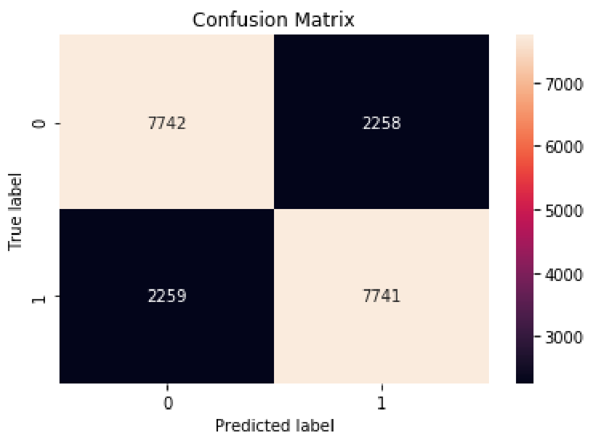
In [96]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s3, rf.predict(X_train_s3)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[96]:

```
Text(0.5,1,'Confusion Matrix')
```



In [97]:

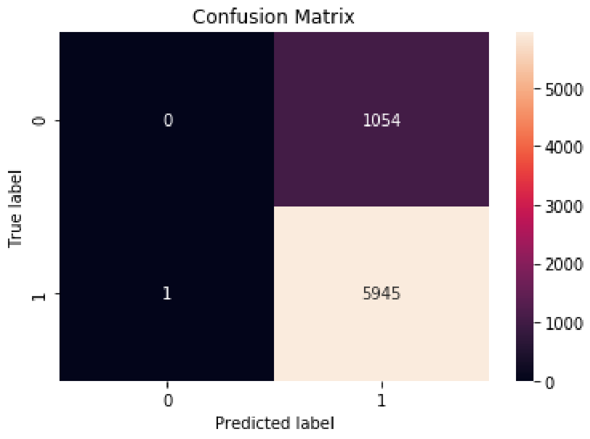
```
# Plotting Plot for Confusion Matrix for Test Data
```

```
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test_s3, rf.predict(X_test_s3)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[97]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.519
- The best paramteres to use with set-3 is max_depth = 4 and n_estimators = 1000
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of Fp and Tp.
- We are getting such a low AUC score because we're selecting only 20k points, if we increase the number of points then the AUC score might increase.
- In test confusion matrix as seen from other method we're not getting high value of FN but instead we are getting high FP.

Applying Random Forest on Set-4 (Tf-Idf W2V)

In [98]:

```
# Selecting 20k points

X_train_s4 = X_train_s4[52113:72113]
y_train_s4 = y_train[52113:72113]

X_test_s4 = X_test_s4[14526:21526]
y_test_s4 = y_test[14526:21526]

print(X_train_s4.shape, y_train_s4.shape)
print(X_test_s4.shape, y_test_s4.shape)
```

```
(20000, 612) (20000,)
(7000, 612) (7000,)
```

In [99]:

```
rf4 = RandomForestClassifier(class_weight = 'balanced', min_samples_split = 10)
parameters = {'n_estimators' : [50, 100, 200, 300, 500, 1000], 'max_depth' : [2, 3, 4, 5, 6, 8, 10]}
clf4 = GridSearchCV(rf4, parameters, cv = 3, scoring = 'roc_auc')
clf4.fit(X_train_s4, y_train_s4)
```

Out[99]:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
```

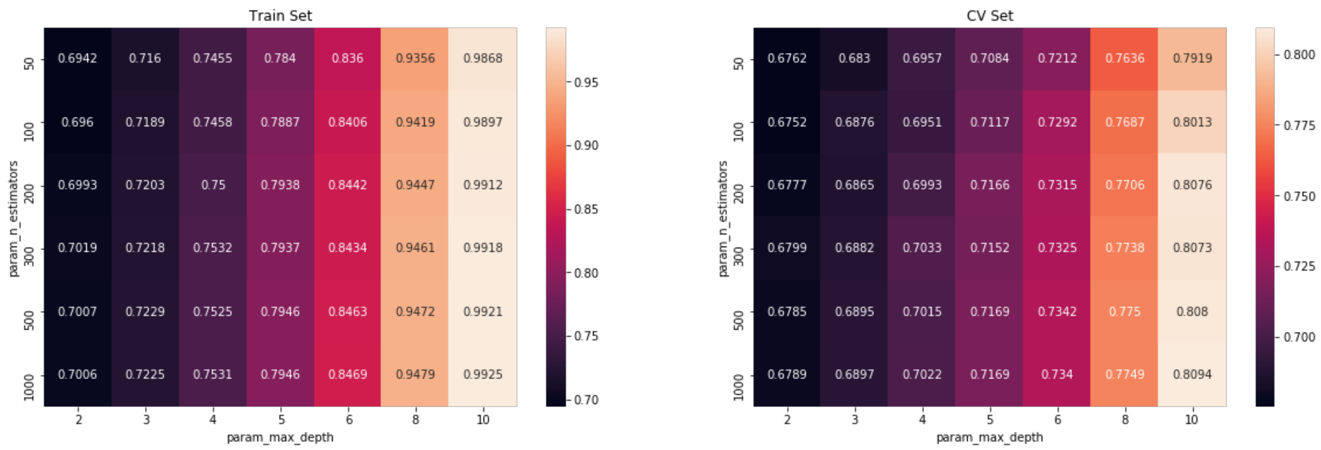
```
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=10, min_weight_fraction_leaf=0.0,
             n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
             verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [50, 100, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 8,
10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Heatmap for Set-4

In [100]:

```
max_scores = pd.DataFrame(clf4.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1, 2, figsize = (20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



param_n_estimators \ param_max_depth (Train Set)	2	3	4	5	6	8	10
50	0.6942	0.716	0.7455	0.784	0.836	0.9356	0.9868
100	0.696	0.7189	0.7458	0.7887	0.8406	0.9419	0.9897
200	0.6993	0.7203	0.75	0.7938	0.8442	0.9447	0.9912
300	0.7019	0.7218	0.7532	0.7937	0.8434	0.9461	0.9918
500	0.7007	0.7229	0.7525	0.7946	0.8463	0.9472	0.9921
1000	0.7006	0.7225	0.7531	0.7946	0.8469	0.9479	0.9925

param_n_estimators \ param_max_depth (CV Set)	2	3	4	5	6	8	10
50	0.6762	0.683	0.6957	0.7084	0.7212	0.7636	0.7919
100	0.6752	0.6876	0.6951	0.7117	0.7292	0.7687	0.8013
200	0.6777	0.6865	0.6993	0.7166	0.7315	0.7706	0.8076
300	0.6799	0.6882	0.7033	0.7152	0.7325	0.7738	0.8073
500	0.6785	0.6895	0.7015	0.7169	0.7342	0.775	0.808
1000	0.6789	0.6897	0.7022	0.7169	0.734	0.7749	0.8094

In [101]:

```
print(clf4.best_estimator_)
print(clf4.score(X_train_s4, y_train_s4))
print(clf4.score(X_test_s4, y_test_s4))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.98531629
0.707587452154782
```

Observations

- Based on the heatmap we can see that values max_depth = 6 and n_estimators = 500 will give us the best results.

Applying Random Forest with Hyperparameter Tuning on Set-4

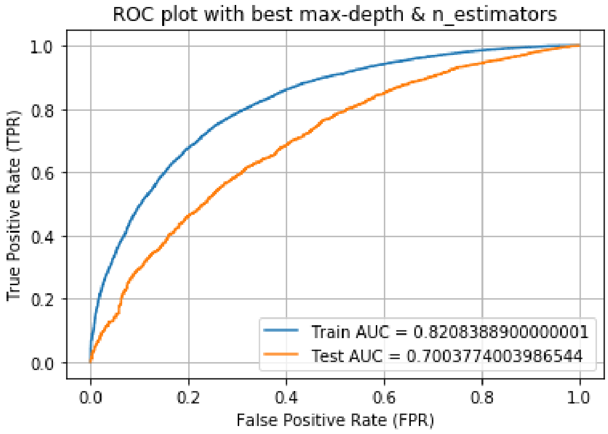
In [102]:

In [102]:

```
rf = RandomForestClassifier(class_weight = 'balanced', max_depth = 6, min_samples_split = 10, n_est
imators = 500)
rf.fit(X_train_s4, y_train_s4)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s4, rf.predict_proba(X_train_s4)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s4, rf.predict_proba(X_test_s4)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

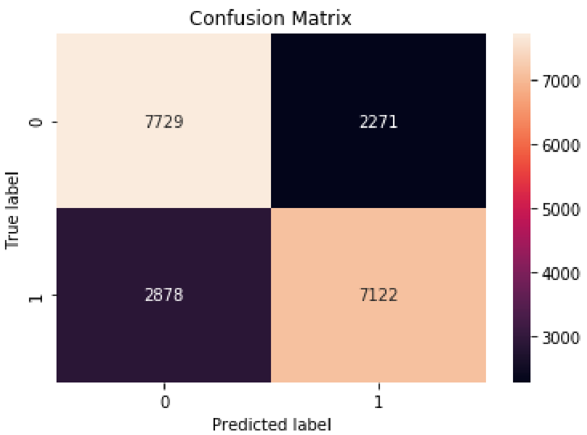
In [103]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s4, rf.predict(X_train_s4)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[103]:

```
Text(0.5,1,'Confusion Matrix')
```



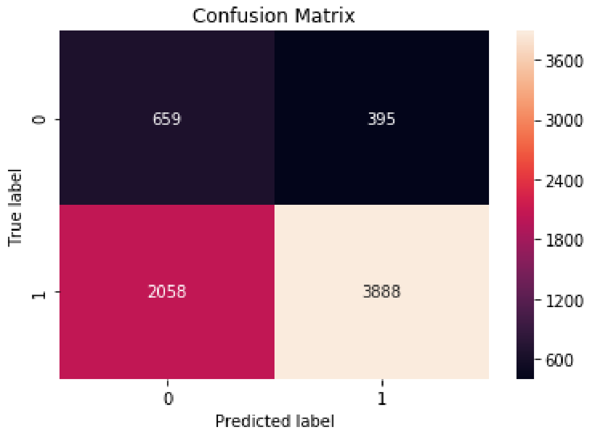
In [104]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()
```

```
sns.heatmap(confusion_matrix(y_test_s4, rf.predict(X_test_s4)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[104]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.700
- The best paramteres to use with set-4 is max_depth = 6 and n_estimators = 500
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.
- Even though we are selecting 20k points the AUC score we got on test data is highest. That is why Tf-Idf W2V is a state of the Art.
- If we have used more number of data points then this AUC will definetly increase.

Gradient Boosting Decision Tree

Applying GBDT on Set-1 (BoW)

In [105]:

```
from sklearn.ensemble import GradientBoostingClassifier

rf5 = GradientBoostingClassifier(min_samples_split = 10)
parameters = {'n_estimators' : [5, 8, 11, 15, 20], 'max_depth' : [2, 3, 5, 7, 10]}
clf5 = GridSearchCV(rf5, parameters, cv = 3, scoring = 'roc_auc')
clf5.fit(X_train_s1, y_train_s1)
```

Out[105]:

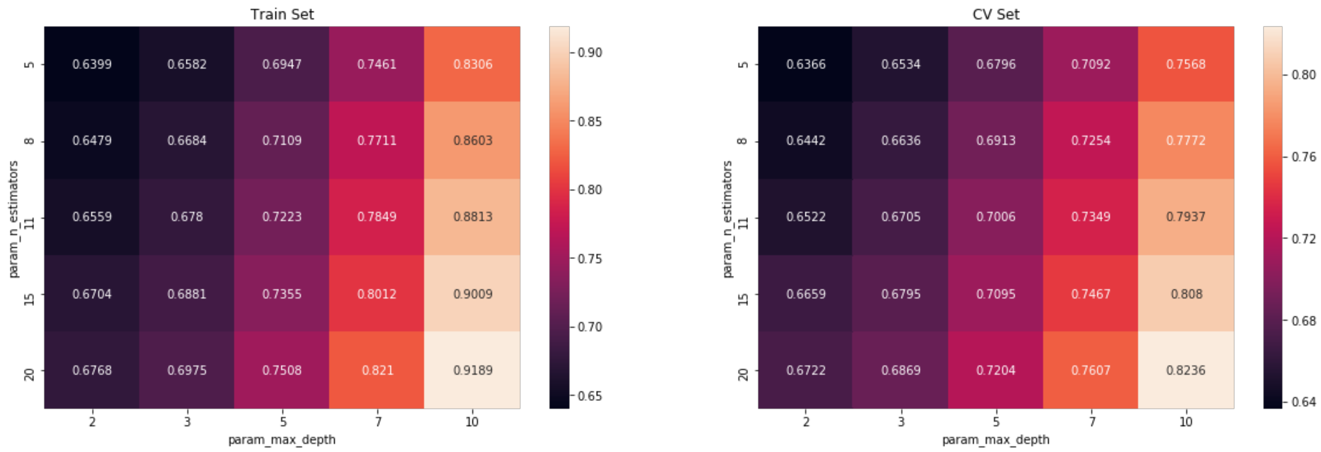
```
GridSearchCV(cv=3, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [5, 8, 11, 15, 20], 'max_depth': [2, 3, 5, 7, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Heatmap for Set-1

In [106]:

```
max_scores = pd.DataFrame(clf5.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [107]:

```
print(clf5.best_estimator_)
print(clf5.score(X_train_s1, y_train_s1))
print(clf5.score(X_test_s1, y_test))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=20,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
0.9038189999999999
0.69902297677764
```

Observations

- Based on the heatmap we can see that values max_depth = 5 and n_estimators = 15 will give us the best results.

Applying GBDT with Hyperparameter Tuning on Set-1

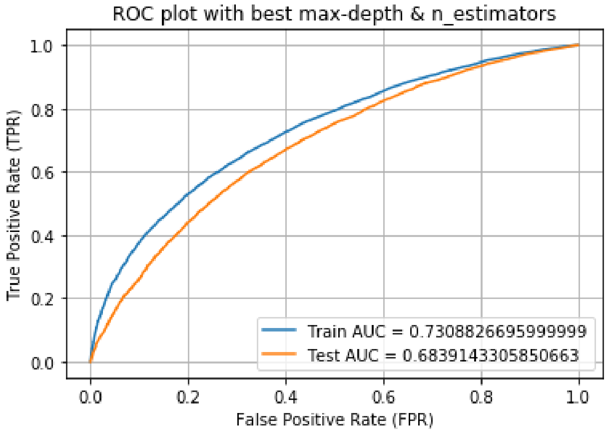
In [108]:

```
rf = GradientBoostingClassifier(max_depth = 5, min_samples_split = 10, n_estimators = 15)
rf.fit(X_train_s1, y_train_s1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s1, rf.predict_proba(X_train_s1)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, rf.predict_proba(X_test_s1)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
```

```
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

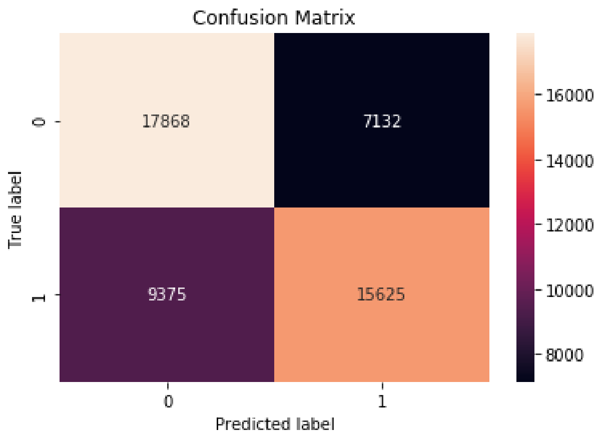
In [109]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s1, rf.predict(X_train_s1)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[109]:

```
Text(0.5,1,'Confusion Matrix')
```



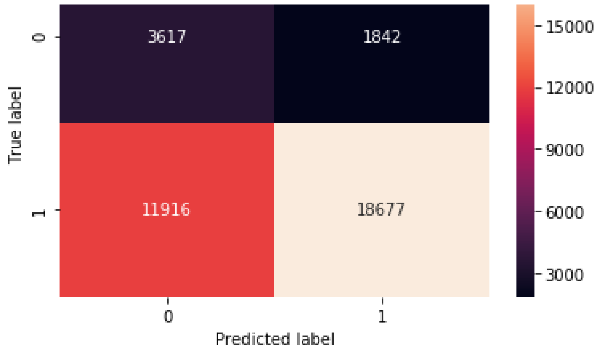
In [110]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, rf.predict(X_test_s1)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[110]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.683
- The best paramteres to use with set-1 is max_depth = 5 and n_estimators = 15
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.

Applying GBDT On Set-2 (Tf-Idf)

In [111]:

```
rf6 = GradientBoostingClassifier(min_samples_split = 10)
parameters = {'n_estimators' : [5, 8, 11, 15, 20], 'max_depth' : [2, 3, 5, 7, 10]}
clf6 = GridSearchCV(rf6, parameters, cv = 3, scoring = 'roc_auc')
clf6.fit(X_train_s2, y_train_s2)
```

Out[111]:

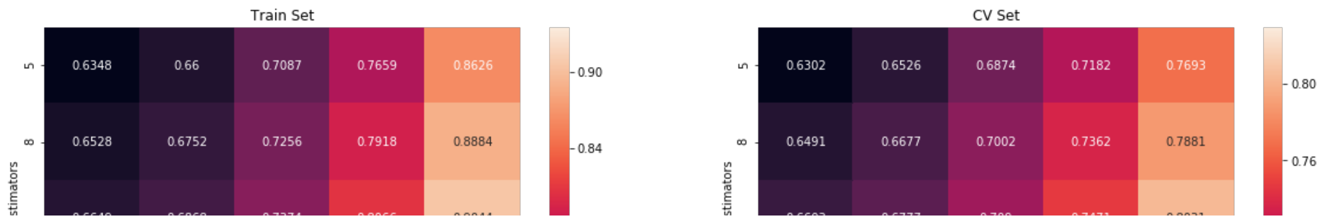
```
GridSearchCV(cv=3, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [5, 8, 11, 15, 20], 'max_depth': [2, 3, 5, 7, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

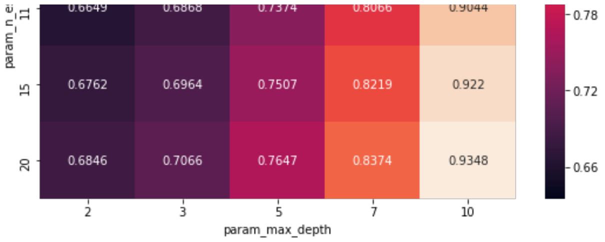
Heatmap for Set-2

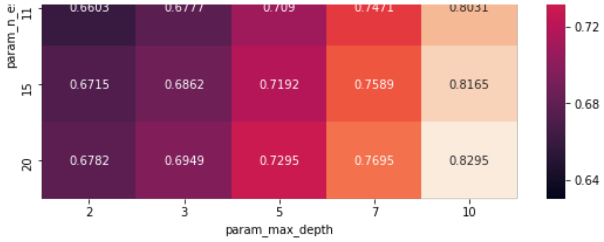
In [112]:

```
max_scores = pd.DataFrame(clf6.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```







In [113]:

```
print(clf6.best_estimator_)
print(clf6.score(X_train_s2, y_train_s2))
print(clf6.score(X_test_s2, y_test))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=20,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
0.9217013568
0.7005044220043057
```

Observations

- Based on the heatmap we can see that values max_depth = 5 and n_estimators = 20 will give us the best results.

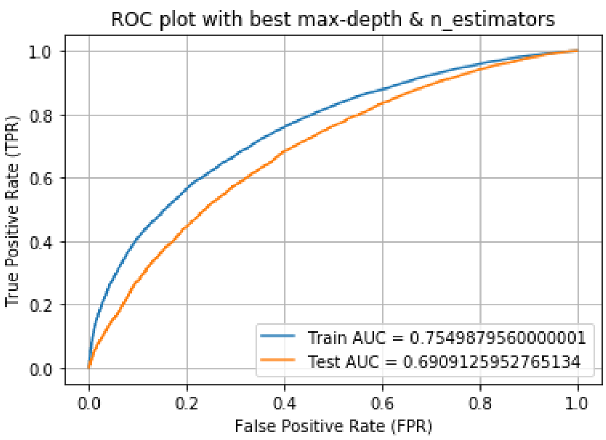
Applying GBDT with Hyperparameter tuning on Set-2

In [114]:

```
rf = GradientBoostingClassifier(max_depth = 5, min_samples_split = 10, n_estimators = 20)
rf.fit(X_train_s2, y_train_s2)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s2, rf.predict_proba(X_train_s2)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, rf.predict_proba(X_test_s2)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

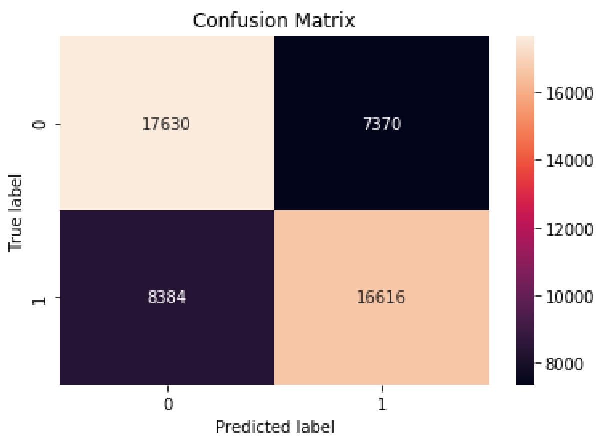
In [115]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s2, rf.predict(X_train_s2)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[115]:

```
Text(0.5,1,'Confusion Matrix')
```



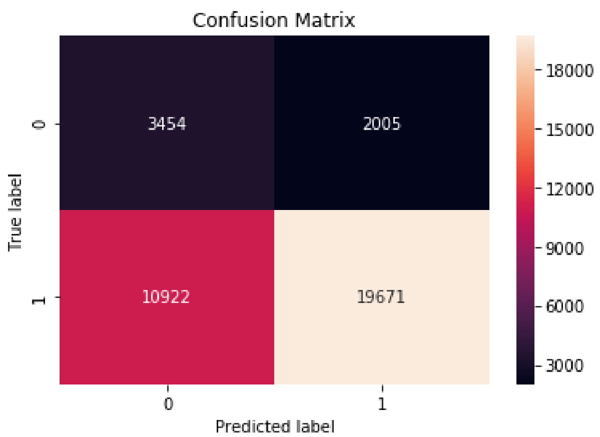
In [116]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test, rf.predict(X_test_s2)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[116]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.690
- The best paramteres to use with set-2 is max_depth = 5 and n_estimators = 20
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.

Applying GBDT on Set-3 (Avg W2V)

In [117]:

```
rf7 = GradientBoostingClassifier(min_samples_split = 10)
parameters = {'n_estimators' : [5, 8, 11, 15, 20], 'max_depth' : [2, 3, 5, 7, 10]}
clf7 = GridSearchCV(rf7, parameters, cv = 3, scoring = 'roc_auc')
clf7.fit(X_train_s3, y_train_s3)
```

Out[117]:

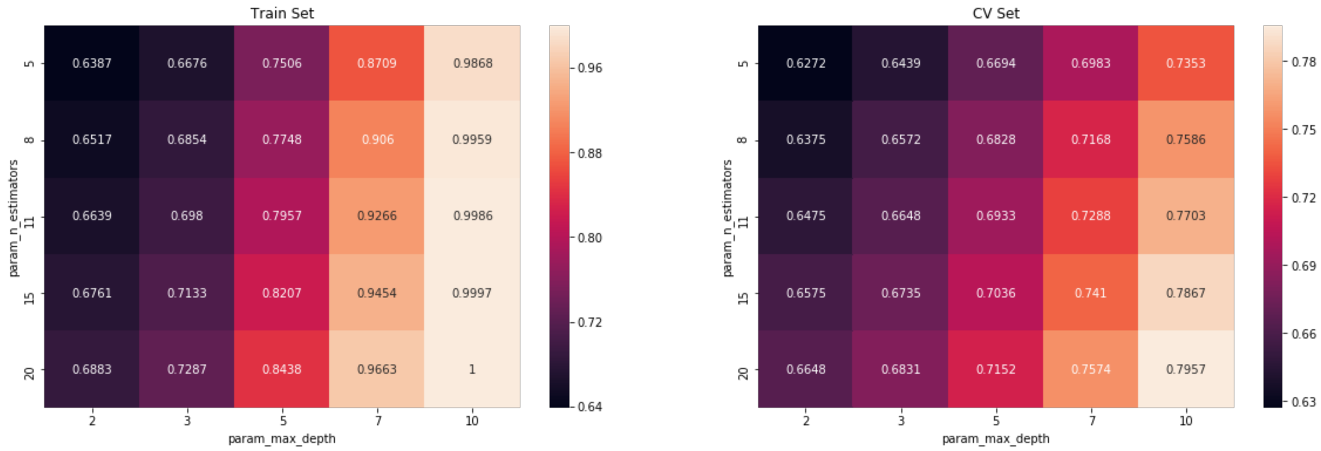
```
GridSearchCV(cv=3, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': [5, 8, 11, 15, 20], 'max_depth': [2, 3, 5, 7, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

Heatmap on Set-3

In [118]:

```
max_scores = pd.DataFrame(clf7.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [119]:

```
print(clf7.best_estimator_)
print(clf7.score(X_train_s3, y_train_s3))
print(clf7.score(X_test_s3, y_test_s3))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=20,
```

```
                presort='auto', random_state=None, subsample=1.0, verbose=0,
                warm_start=False)
0.9995916
0.5278489964391734
```

Observations

- Based on the heatmap we can see that values max_depth = 3 and n_estimators = 15 will give us the best results.

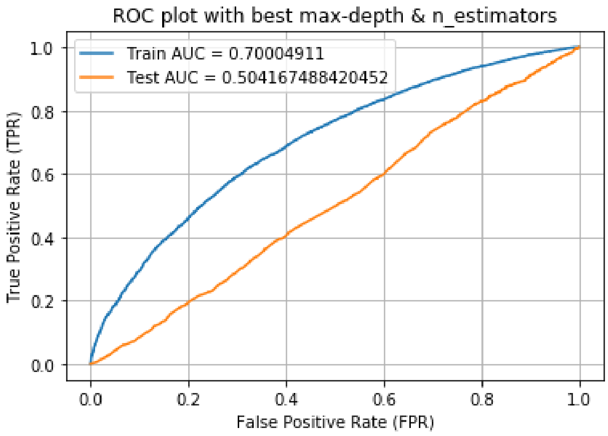
Applying GBDT with Hyperparameter Tuning on Set-3

In [130]:

```
rf = GradientBoostingClassifier(max_depth = 3, min_samples_split = 10, n_estimators = 15)
rf.fit(X_train_s3, y_train_s3)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s3, rf.predict_proba(X_train_s3)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s3, rf.predict_proba(X_test_s3)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

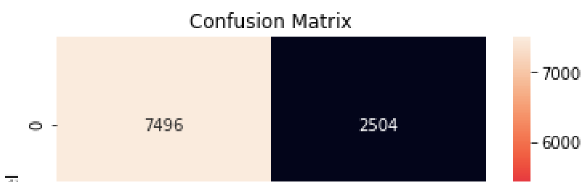
In [121]:

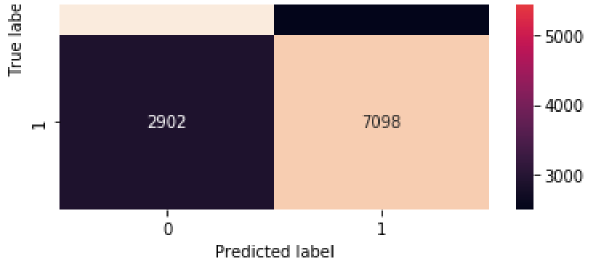
```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s3, rf.predict(X_train_s3)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[121]:

```
Text(0.5,1,'Confusion Matrix')
```





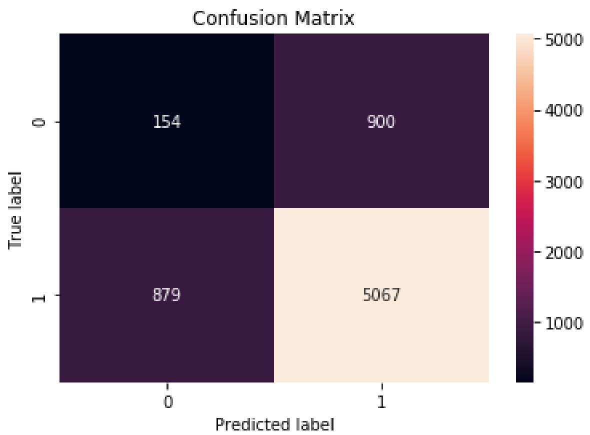
In [122]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test_s3, rf.predict(X_test_s3)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[122]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.504
- The best paramteres to use with set-3 is max_depth = 3 and n_estimators = 15
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of Fp and Tp.
- We are getting low AUC score because we are selecting only 20k points.

Applying GBDT on Set-4 (Tf-Idf W2V)

In [123]:

```
rf8 = GradientBoostingClassifier(min_samples_split = 10)
parameters = {'n_estimators' : [5, 8, 11, 15, 20], 'max_depth' : [2, 3, 5, 7, 10]}
clf8 = GridSearchCV(rf8, parameters, cv = 3, scoring = 'roc_auc')
clf8.fit(X_train_s4, y_train_s4)
```

Out[123]:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
```

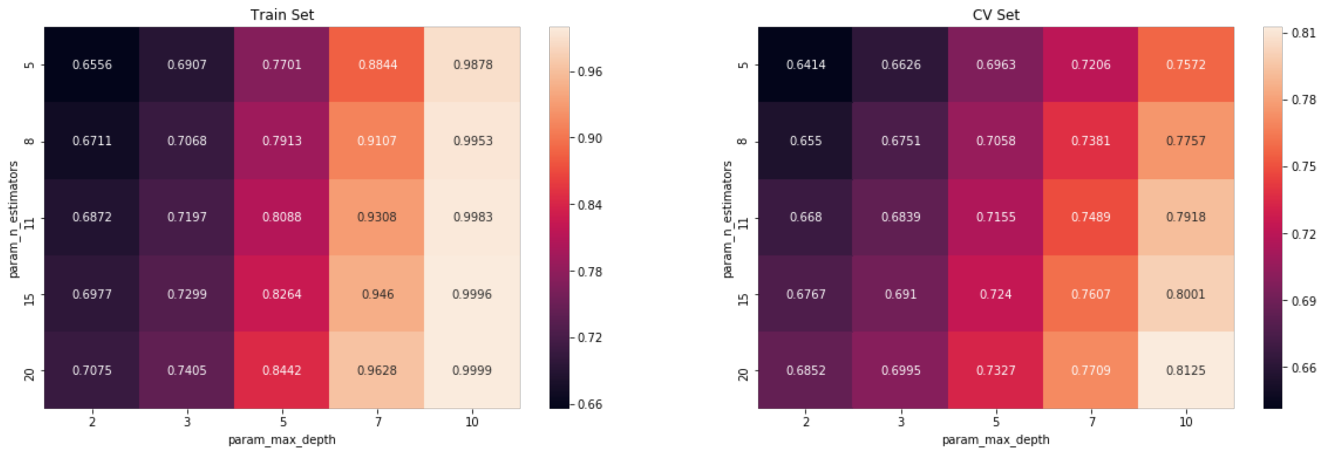
```
        warm_start=False),
    fit_params=None, iid=True, n_jobs=1,
    param_grid={'n_estimators': [5, 8, 11, 15, 20], 'max_depth': [2, 3, 5, 7, 10]},
    pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
    scoring='roc_auc', verbose=0)
```

Heatmap for Set-4

In [124]:

```
max_scores = pd.DataFrame(clf8.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [125]:

```
print(clf8.best_estimator_)
print(clf8.score(X_train_s4, y_train_s4))
print(clf8.score(X_test_s4, y_test_s4))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=20,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
0.9994308449999999
0.6820609074331858
```

Observations

- Based on the heatmap we can see that values max_depth = 5 and n_estimators = 20 will give us the best results.

Applying GBDT with Hyperparameter tuning on Set-4

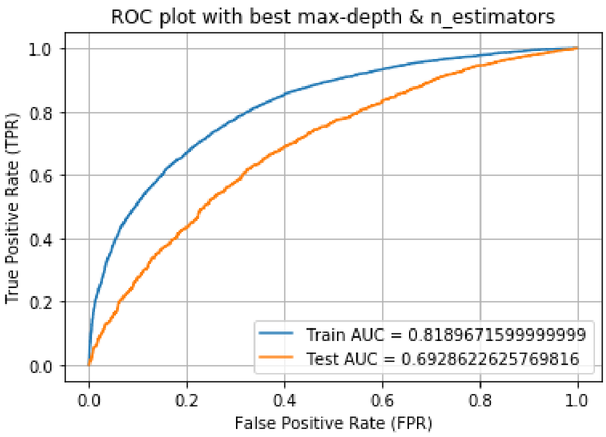
In [126]:

```
rf = GradientBoostingClassifier(max_depth = 5, min_samples_split = 10, n_estimators = 20)
rf.fit(X_train_s4, y_train_s4)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_s4, rf.predict_proba(X_train_s4)[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s4, rf.predict_proba(X_test_s4)[:, 1])
```

```
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_s4, rf.predict_proba(X_test_s4)[:, 1])

plt.plot(train_fpr, train_tpr, label="Train AUC = " + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC = " + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate (TPR)")
plt.xlabel("False Positive Rate (FPR)")
plt.title("ROC plot with best max-depth & n_estimators")
plt.grid()
plt.show()
```



Plotting Confusion Matrix for Train and Test data

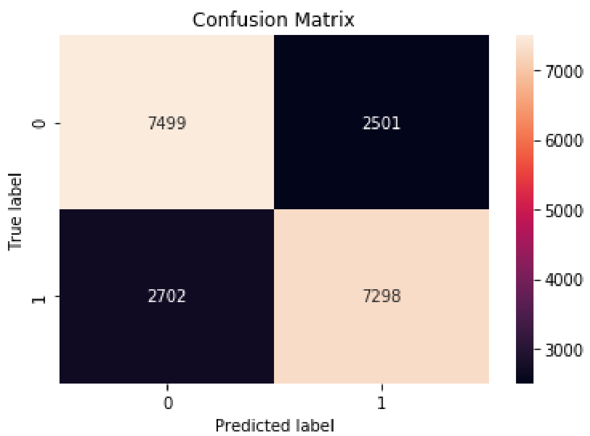
In [127]:

```
# Plotting Plot for Confusion Matrix for train data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_train_s4, rf.predict(X_train_s4)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[127]:

```
Text(0.5,1,'Confusion Matrix')
```



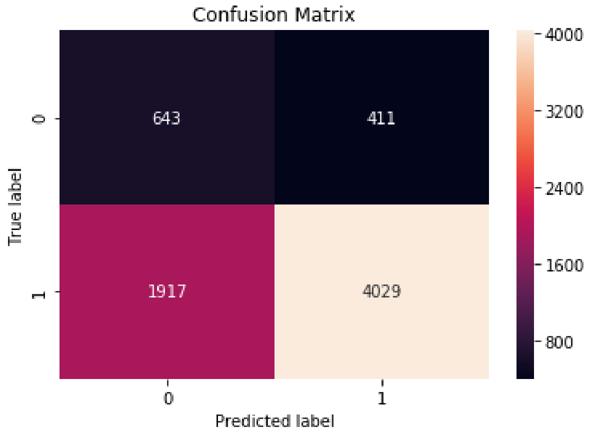
In [128]:

```
# Plotting Plot for Confusion Matrix for Test Data
ax = plt.subplot()

sns.heatmap(confusion_matrix(y_test_s4, rf.predict(X_test_s4)), annot=True, ax = ax, fmt='g')
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.set_title('Confusion Matrix')
```

Out[128]:

```
Text(0.5,1,'Confusion Matrix')
```



Observations

- The AUC score we got on Test data is 0.692
- The best paramteres to use with set-4 is max_depth = 5 and n_estimators = 20
- Confusion matrix of train data is good as values of TN and TP is high.
- In confusion matrix of test data we are getting high values of FN and Tp.
- We are using only 20k points then also we are getting such a high AUC score.

In [132]:

```
from prettytable import PrettyTable

t = PrettyTable()
t.field_names= ("Vectorizer", "Model","max_depth" , "n_estimators", "AUC")
t.add_row(["BoW", "Random Forest", 6, 500, 0.681])
t.add_row(["Tf-Idf", "Random Forest", 6, 500, 0.687])
t.add_row(["Avg_W2V", "Random Forest", 4, 1000, 0.519])
t.add_row(["Tf-Idf_W2V", "Random Forest", 6, 500, 0.700])
t.add_row(["BoW", "GBDT", 5, 15, 0.683])
t.add_row(["Tf-Idf", "GBDT", 5, 20, 0.690])
t.add_row(["Avg_W2V", "GBDT", 3, 15, 0.504])
t.add_row(["Tf-Idf_W2V", "GBDT", 5, 20, 0.692])

print(t)
```

```
+------------+---------------+-----------+--------------+-------+
| Vectorizer |     Model     | max_depth | n_estimators |  AUC  |
+------------+---------------+-----------+--------------+-------+
|    BoW     | Random Forest |     6     |     500      | 0.681 |
|   Tf-Idf   | Random Forest |     6     |     500      | 0.687 |
|  Avg_W2V   | Random Forest |     4     |     1000     | 0.519 |
| Tf-Idf_W2V | Random Forest |     6     |     500      |  0.7  |
|    BoW     |      GBDT     |     5     |      15      | 0.683 |
|   Tf-Idf   |      GBDT     |     5     |      20      |  0.69 |
|  Avg_W2V   |      GBDT     |     3     |      15      | 0.504 |
| Tf-Idf_W2V |      GBDT     |     5     |      20      | 0.692 |
+------------+---------------+-----------+--------------+-------+
```

Conclusion

- Even though in Tf-Idf W2V we have only used 20k points we are getting highest AUC score.
- The highest AUC score we have achieved on Random Forest is 0.7 using Tf-Idf W2V.
- The highest AUC score we have achieved on GBDT is 0.692 using Tf-Idf W2V.
- In Avg-W2V also we are selecting 20k points and getting low AUC scores. If we have used more datapoints then we might get AUC scores better than this.
- AUC scores that we have achieved by Random Forest and GBDT are nearly same.
- AUC scores we have achieved on all these sets by RF and GBDT are best as compare to other models we have built

- AUC scores we have achieved on all these sets by RF and GBDT are best as compare to other models we have built.
- The run-time of Random forest and GBDT is too high.